Research Report 310

Advances in predictive wavelet transform image coding

Ian Levy
Roland Wilson

RR310

In this work, it is shown that fractal image coding can be placed in a conventional predictive framework by using a wavelet representation. This has the significant advantages of simplifying decoding by avoiding iteration and of relating the technique to traditional coding methods. The resulting coder is a combination of transform and predictive techniques. A novel frequency domain subblock search is presented along with a tree structured vector quantizer. Some results are presented for a variety of images and bit-rates followed by some speculation on further research.

Department of Computer Science
University of Warwick
Coventry CV4 7AL
United Kingdom

July 1996

Advances in predictive wavelet transform image coding

Ian Levy, Roland Wilson
Department of Computer Science,
University of Warwick,
Coventry

July 30, 1996

Abstract

In this work, it is shown that fractal image coding can be placed in a conventional predictive framework by using a wavelet representation. This has the significant advantages of simplifying decoding by avoiding iteration and of relating the technique to traditional coding methods. The resulting coder is a combination of transform and predictive techniques. A novel frequency domain subblock search is presented along with a tree structured vector quantizer. Some results are presented for a variety of images and bit-rates followed by some speculation on further research.

Contents

1 Introduction 1

2 The New Algorithm 2
- 2.1 Introduction . . . 2
- 2.2 Basic Algorithm . . . 2
- 2.3 The Block Search Algorithm . . . 3
- 2.4 The Karhunen-Loève transform . . . 5

3 The Tree Structured Vector Quantizer 6
- 3.1 Introduction . . . 6
- 3.2 Building a Tree Structured Vector Quantizer . . . 7
- 3.3 The Linde-Buzo-Gray Vector Quantizer Design Algorithm . . . 8
- 3.4 Restructuring an Unbalanced Tree . . . 9

4 Coder Results 12

5 Tree Structured Vector Quantizer Results 16

6 Discussion,Conclusions and Further Work 27

References 28

List of Figures

1 Coder block diagram 4
2 Two clusters, the centroids (shaded) and their separating hyperplane 6
3 Incremental growing of a tree structured vector quantizer 8
4 A tree generated from artificial data 10
5 The tree restructured from the root 11
6 Reconstruction of Lena with no search 12
7 Reconstruction of Lena with full search 13
8 Reconstruction of Boats with no search 13
9 Reconstruction of Boats with full search 14
10 JPEG Reconstruction at 0.22 bpp, PSNR 30.71 15
11 Results for data in the training set $\sigma^2 = 1000$ 20
12 Results for data outside training set $\sigma^2 = 1000$ 21
13 Results for data in the training set $\sigma^2 = 200$ 25
14 Results for data outside training set $\sigma^2 = 200$ 26

1 Introduction

In recent years, both fractal coding and wavelet transform coding algorithms have been the subject of extensive research [1] [7] [4] [11] [14]. Although they share a reliance on the scaling properties of the edge features which dominate normal images, there has previously been little attempt to marry them (see [13] however).

In a wavelet decomposition, the image is recursively decomposed into frequency bands. The wavelet subbands contain the high frequency components of the image in the horizontal, vertical and diagonal directions. These components can easily be vector quantized with low visible distortion because (it is argued) the wavelets are better matched to edge features than the sinusoidal basis functions used in earlier transform coders. Fractal image coding relies on the concepts of self-similarity and affine maps, which may be used to describe how to transform one image region into the other. The image is tiled with such regions and the collection of the affine maps which results is known as the *fractal code* for the image. An approximation to the original image is reconstructed from the fractal code by iterating the affine maps on any initial image [7] [4] [3].

Recently, there have been several attempts to combine fractal and transform compression methods [13] [5] [6]. These have, for the most part, used DCT basis vectors to represent the error resulting from the fractal approximation. There have been few advances in searching for fractal based systems. In this paper, a novel search method is presented which takes into account the isometry group used.

Also, a tree structured vector quantizer is presented, based on the LBG algorithm. This structure also embodies a rate-distortion measure, allowing variable rate encoding using the same data. This structure will be used to address domain blocks in near-optimal number of bits.

2 The New Algorithm

2.1 Introduction

The new method is largely based on the coder presented in [8] and [9]. It can be seen as a predictive coder, in which blocks of data representing larger scales are used to predict those at smaller scales. Errors are coded using a Karhunen-Loève transform coder operating in the wavelet domain.

2.2 Basic Algorithm

The novel algorithm presented combines parts of both wavelet and fractal coding. The image to be coded is decomposed by the wavelet transform to a given level, say l, known as the *base* level. The first level to be approximated, called the *domain level*, is then decided. This is any level below the base level. The *range level* is then the level from which the prediction occurs (usually the level above the domain level). Levels from l to the range level are quantized using a linear quantizer for the highpass coefficients and a separate quantizer for the lowpass coefficients at the highest level. The quantized approximation is used as the range level of coefficients to reduce errors at the decoder. Each of the bands in the domain level is partitioned into non-overlapping domain blocks. Let $\{D_i^{HL}\}$, $\{D_i^{LH}\}$ and $\{D_i^{HH}\}$ be the domain blocks from each subband. For each domain block set $\{D_i^{HL}, D_i^{LH}, D_i^{HH}\}$, a corresponding range block set $\{R_i^{HL}, R_i^{LH}, R_i^{HH}\}$ must be found which minimizes the error

$$\epsilon = \|D_i^{HL} - \hat{D}_i^{HL}\|^2 + \|D_i^{LH} - \hat{D}_i^{LH}\|^2 + \|D_i^{HH} - \hat{D}_i^{HH}\|^2 \tag{1}$$

where $\hat{D}_i^X = \iota_i(\alpha_i \cdot \hat{R}_i^X)$, $\hat{R}_i$ is a normalised range block from sub-band X,

$$\alpha_i = \frac{< \hat{R}_i^{HH}, D_i^{HH} > + < \hat{R}_i^{HL}, D_i^{HL} > + < \hat{R}_i^{LH}, D_i^{LH} >}{< \hat{R}_i^{HH}, \hat{R}_i^{HH} > + < \hat{R}_i^{HL}, \hat{R}_i^{HL} > + < \hat{R}_i^{LH}, \hat{R}_i^{LH} >} \tag{2}$$

and $\{\iota_i\}_{i=0}^7$ is the dihedral group of the square. Hence, the blocks from the three subbands are scaled by the same factor.

Since an orthogonal wavelet transform is used, the subbands are all orthogonal. Hence, no iteration of the affine maps is required. Since there will be no iteration of the maps in the decoding process, the value of α_i is unconstrained. Given a domain block size d and and search radius r, we define the *search region* as the subset of the subband $\mathcal{I}$ given by $([x-rd, x+rd] \times [y-rd, y+rd]) \cap \mathcal{I}$. A search within this region is performed using the method described in section 2.3. This method constrains the maps so that each range block R_i^X ($X \in \{HH, HL, LH\}$) is in the same relative position, but in different subbands. The reasoning behind this is that if two blocks are similar, then their horizontal, vertical and diagonal components must also be similar.

This process is then repeated on successive levels of the wavelet decomposition. The approximation of the domain level generated by the coder thus becomes the range coefficients for the next iteration. Separate affine map coefficients are generated between levels, up to level 1. As it stands, the coder attempts to extrapolate a block in the range level to a block in the domain level in each subband. However, if there is not a good match between wavelet levels, the overall reconstruction error will increase. To overcome this, block projection is performed after the fractal coding on each level to reduce errors, in the same vein as Huang and Gharavi-Alkhansari [6]. The present method, however, is simpler, since it is based on the Karhunen-Loève transform for the symmetrised error vectors. A block diagram of the coder is shown in figure 1. It is worth noting that the coder can operate in a variety of modes, depending on the application. It is possible to vary the block size, enable or disable the Karhunen-Loève transform sub-system and enable or disable the block searching (ie restrict the search region to a single block). Obviously, different configurations will be suitable for different applications, each having different bit rate and error requirements.

The parameter sets that describe the blockwise maps and the basis projection coefficients are linearly quantized and encoded using an order zero arithmetic coder. A *rate value* determines how many bins the quantizer uses and how many symbols the arithmetic coder can encode. Reconstruction proceeds much the same as for standard fractal coding. The major difference is that this method does not require iteration of the maps. If the Karhunen-Loève transform sub-system is enabled, then the projected error vectors are added to the reconstructed approximation of the block. An inverse wavelet transform from the base level reconstructs the original image.

2.3 The Block Search Algorithm

Most, if not all, fractal based coders use a simple search on the domain pool. These searches usually proceed by extracting a domain block from the pool and applying each isometry to the domain block to find which orientation of the block most closely matches the range block. This is repeated over all blocks in the domain pool to find the best in the least-squares sense. This has obvious disadvantages in both computational complexity and the time required to perform the search.

The method to be presented here is a Fourier-domain search. Define $\mathcal{F}(x)$ to be the Fourier transform of x, x^* to be the complex conjugate of x and $\mathcal{I}(x)$ to be the inverse Fourier transform of x. Given an $n \times n$ range block R and a normalised domain block D, the domain block is padded with zeros to $n \times n$. A correction factor matrix, C, is generated from the range block by

$$C_{i,j} = \frac{\sum_{k=0}^{n} \sum_{l=0}^{n} R^2_{(i+k),(j+l)}}{n^2} \tag{3}$$

C represents the energy in each sub-block in the range block.

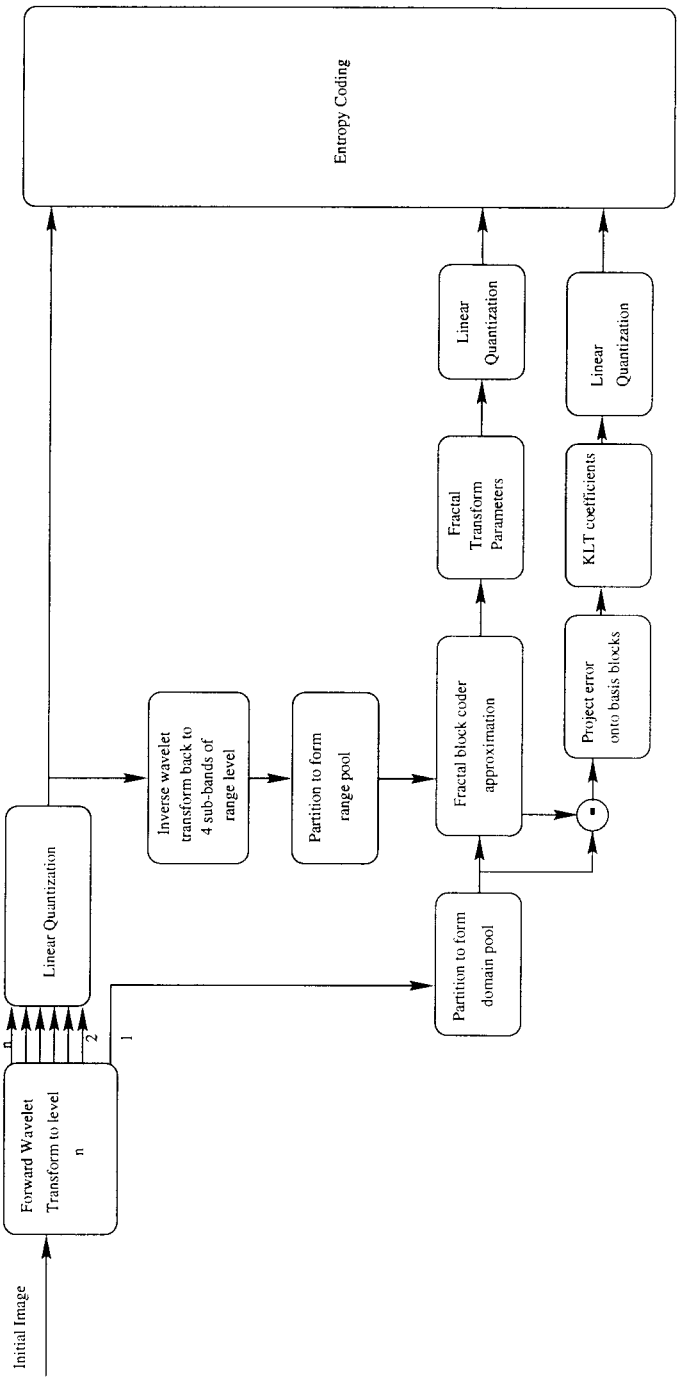


Figure 1: Coder block diagram

The correlation matrix between the two blocks is generated by

$$\rho = Re\left(\mathcal{I}\left(\mathcal{F}(R)\mathcal{F}^{*}(D)\right)\right) \tag{4}$$

Then, the normalised correlation between the two blocks is simply defined as

$$\hat{\rho}_{i,j} = \frac{\rho_{i,j}}{C_{i,j}} \tag{5}$$

The position of the highest value in $\hat{\rho}$ corresponds to the origin of the best matching block. The block isometries are applied to the padded domain block transform. If the isometry contains a rotation of π radians, the transform is simply conjugated.

The search is actually performed across all three bands simultaneously. The $\hat{\rho}$ matrices derived from each band are added up and the highest value found. This is then the origin of the three blocks that match the domain blocks the best.

2.4 The Karhunen-Loève transform

The Karhunen-Loève transform [1] is well known in the field of image processing. It will be described briefly here for completeness. Given a sample from a population of vectors $\{x_i\}$ of the form $x_i = \{x_{i_1}, \ldots, x_{i_n}\}$, the *covariance matrix* is estimated using the sample mean

$$C_x = \frac{1}{M}\sum_{k=1}^{M} x_k x_k^T - \bar{x}\bar{x}^T \tag{6}$$

where $\bar{x}$ is the mean of the sample, defined as

$$\bar{x} = \frac{1}{M}\sum_{k=1}^{M} x_k. \tag{7}$$

The element c_{ij} of C_x is the covariance of the elements x_i and x_j. Hence, the matrix C_x is real and symmetric. Therefore, it is always possible to find a set of real eigenvectors of C_x. It is these eigenvectors that are used as the basis vectors for error correction in the coder. That is, any residual error in the coder after the wavelet prediction is projected onto these basis vectors and the coefficients transmitted. The actual vectors used are calculated over a variety of images and are known to both the encoder and decoder.

[1] Also known as the Hotelling, eigenvector or principal component transform

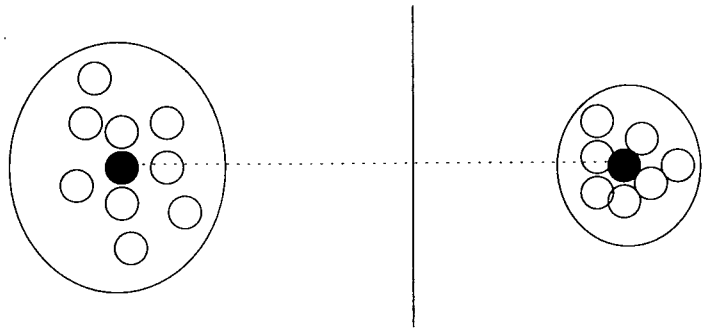


Figure 2: Two clusters, the centroids (shaded) and their separating hyperplane

3 The Tree Structured Vector Quantizer

3.1 Introduction

Even with the frequency domain block search, extracting each range block in the search region and testing it is still slow. It would obviously speed the search if only *similar* blocks were tested. Also, building a history of previous blocks is bound to reduce error rate, since the coder is given a more richly populated vector space to predict from. Vector Quantization is a common method of providing lossy data compression. The vector quantizer works by providing a *codebook* of vectors which is used to approximate input vectors. The main problem with general VQ methods is that they are highly compute intensive. One method of overcoming this is to structure the quantizer. A particular example of this subclass is the tree structured vector quantizer. There have been some attempts to generate Tree Structured Vector Quantizers [12] [2]. These, however, work in an off-line manner (all the data must be present when the tree is built) or impose some condition on the source. However, for this application, an incremental method is required with no limitations.

Consider a set of data in the Euclidean n-space, $\{x_i : i = 0, \ldots, n-1\} \in \Re^n$. Assuming $n \geq 2$, it is possible to partition the data into two optimal clusters, C_1 and C_2, separated by a decision hyperplane. The situation is shown in Figure 2.

The decision hyperplane is defined by

$$H = \{x_i \in \Re^n : \|x_i - \mu_1\| = \|x_i - \mu_2\|\} \tag{8}$$

where μ_1 and μ_2 are the centroids of the clusters. Expanding this condition gives us

$$\begin{aligned} \|x_i - \mu_1\| &= \|x_i - \mu_2\| \\ &\Downarrow \\ x_i^T x_i - 2x_i^T \mu_1 + \mu_1^T \mu_1 &= x_i^T x_i - 2x_i^T \mu_2 + \mu_2^T \\ &\Downarrow \\ 2x_i^T(\mu_2 - \mu_1) &= \mu_2^T \mu_2 - \mu_1^T \mu_1 \end{aligned} \tag{9}$$

Hence, $\mu_2 - \mu_1$ is the direction of the decision hyperplane normal and $\mu_2^T\mu_2 - \mu_1^T\mu_1$ is the distance along the normal to the point of intersection with the hyperplane itself.

Given a new vector,x, it is a simple to decide which cluster the vector should lie in, without calculating the mean square error between the new vector and each of the cluster centroids. All that is required is to calculate the value of $2x^T(\mu_2 - \mu_1)$. Obviously,

$$\begin{array}{l} 2x^T(\mu_2 - \mu_1) < \mu_2^T\mu_2 - \mu_1^T\mu_1 \quad \text{if } x \in C_1 \\ 2x^T(\mu_2 - \mu_1) > \mu_2^T\mu_2 - \mu_1^T\mu_1 \quad \text{if } x \in C_2 \end{array} \tag{10}$$

with an arbitrary choice being made if equality occurs.

3.2 Building a Tree Structured Vector Quantizer

Each node in the tree will contain the following information

- Pointers to the left son, right son and parent nodes
- The data vector associated with this node
- The direction of the separating hyperplane normal vector and intersection point
- The average distortion between all leaf data and the data vector at this node
- A usage count which indicates the number of leaves below this node

When adding a vector to an existing tree, the following method is applied. At each node, a dichotomy must be evaluated to decide which path to traverse. It requires a simple inner product calculation. The decision is made in the same way as that of equation 10. All dichotomies down a particular path are evaluated until a leaf node is reached. As each node is passed, its average distortion,$\bar{d}_C$, updated by equation 11 and the usage count,n_x, is then incremented.

$$\bar{d}_C = \frac{(\bar{d}_C \times n_x) + \|x - \eta\|^2}{n_x + 1} \tag{11}$$

Once a leaf node has been reached, a comparison is made to see if the data to be added is already present in the leaf. If it is, the usage count of the leaf in incremented and no more is done. Otherwise, two child nodes are spawned from the old leaf node. One of these has the data from the old leaf copied down and the other has the new data placed in it. The average distortion between the old leaf node and the two new children is then calculated and stored in the old leaf node, followed by the hyperplane normal direction and intersection point. It is worth noting explicitly that if the old leaf has a usage count greater than one, this is copied down to the duplicate child node. The incremental growing of the tree is depicted in figure 3.

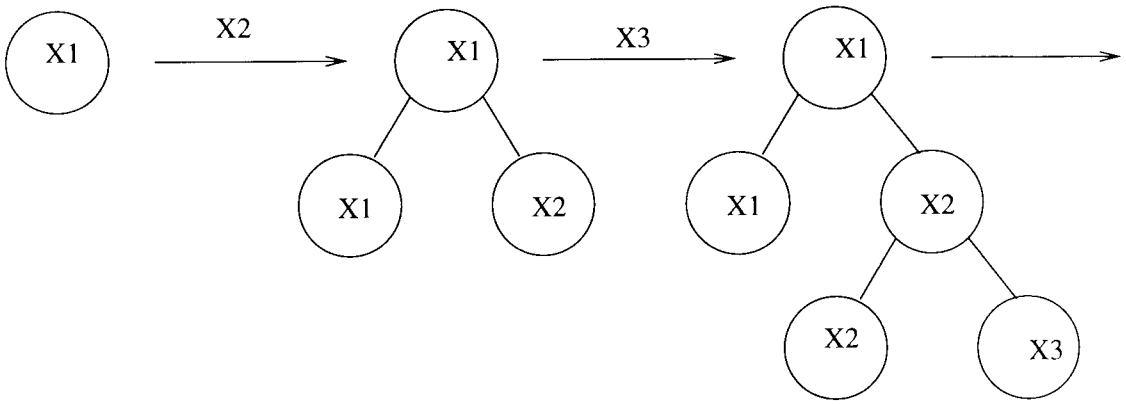


Figure 3: Incremental growing of a tree structured vector quantizer

Since the separating hyperplanes are fixed by their placement between the first 2 vectors, it is quite possible, if not likely, that the tree will become unbalanced at any point below the root. To combat this, a method for restructuring subtrees has been designed, based on the Linde-Buzo-Gray vector quantizer design algorithm, detailed next.

3.3 The Linde-Buzo-Gray Vector Quantizer Design Algorithm

The LBG algorithm for vector quatizer design [10] is used to design near-optimal block quantizers with a certain number of representative vectors. The quantizer is *trained* by a sequence of training data, usually taken to be representative of the population that the quantizer must encode. The algorithm produces an optimum partition of the training set. This is simply extended into an n-dimensional optimum partition of Euclidean n-space by the nearest neighbour rule.

Given N, the number of representative vectors in the final quantizer, a training sequence $\{x_j : j = 0, \ldots, n-1\}$, and an error threshold $\epsilon \geq 0$, the algorithm proceeds as follows.

- Initialisation
  - Given $\{x_j\}$, set $\hat{A}_0(1) = \bar{x}$ where $\bar{x}$ is the centroid of the training sequence.
  - Given $\hat{A}_0(M)$, containing M vectors, perturb each vector in the set by a small, fixed vector, ν. So, each $y_i \in \hat{A}_0(M)$ is replaced by the two vectors $\{y_i + \nu, y_i - \nu\} \in \hat{A}_0(2M)$. Set M=2M.
  - Repeat previous step until $M = N$. Then, the resultant set $\hat{A}_0(N)$ is the initial set to which the partitioning algorithm is applied.
- Partitioning

- Given $\hat{A}_m$, find the minimum distortion partition $\mathcal{P}(\hat{A}_m) = \{C_i : i = 1, \ldots, N\}$ of the training set by assigning each vector x_j to a cluster C_i using the rule

$$x_j \in C_i \Leftrightarrow \|x_j - y_i\| \leq \|x_j - y_l\| \forall l. \tag{12}$$

- Calculate the average distortion

$$D_m = \frac{1}{n} \sum_{j=0}^{n-1} \min_{y \in \hat{A}_m} \|x_j, y\|. \tag{13}$$

If $\frac{D_{m-1} - D_m}{D_m} \leq \epsilon$ stop with $\hat{A}_m$ as the final set of representative vectors.

- Re-estimate the centroids of the clusters $\{C_i\}$. Set $\hat{A}_{m+1} = \{\bar{C}_i : i = 1, \ldots, N\}$. Set $m = m + 1$. Repeat the partitioning step.

3.4 Restructuring an Unbalanced Tree

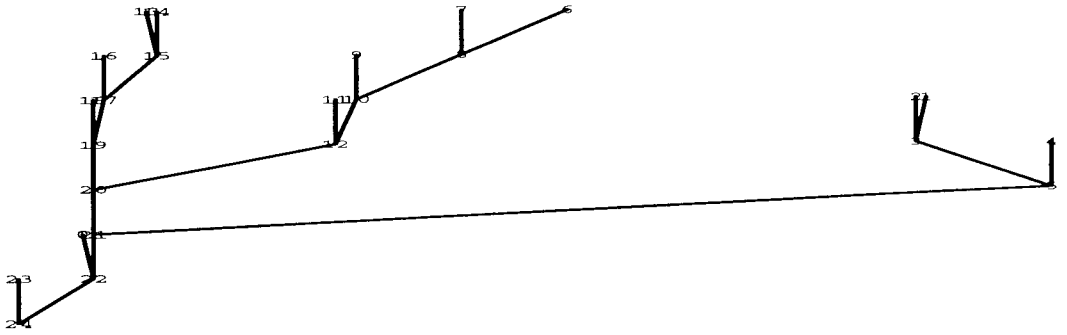
Before a restructure is performed, it must first be decided where (or if) the tree is actually unbalanced. Since the usage count in any node represents the number of leaf data nodes below it, a comparison of usage counts of sibling nodes is an appropriate measure of how unbalanced the subtree is. Note that the root node usage count is the total number of leaf node data vectors in the subtree.

A recursive traversal down the tree is performed from the root. At any point, $n_{Root}, n_{Left}, n_{Right}$, the usage counts in the subtree root node and the current left and right child nodes, are known and it is simple to calculate p_R and p_L thusly

$$p_R = \log_2 \left(\frac{n_{Root}}{n_{Left}} \right) \tag{14}$$

$$p_L = \log_2 \left(\frac{n_{Root}}{n_{Right}} \right) \tag{15}$$

A subtree is deemed unbalanced if $|p_R - p_L| > 0.5$. Once an unbalanced subtree is found, the leaf data is extracted and the subtree is destroyed. The subtree root data value is set to the centroid of all the leaf data. Then, the LBG algorithm is run on the leaf data to produce two cluster centroids. These centroids are used as the data for the child nodes of the subtree root node. The separating hyperplane is calculated and its normal vector and intersection point are stored in the subtree root. The application of the LBG algorithm implicitly generates two new subtree root nodes and two sets of leaf data nodes. These are the two sets of data that constitute the clusters generated by the algorithm. The procedure is applied recursively until only 2 data vectors remain. Then, a simple split is performed and the subtree is fully restructured. The final procedure in the restructure is to work back up the subtree

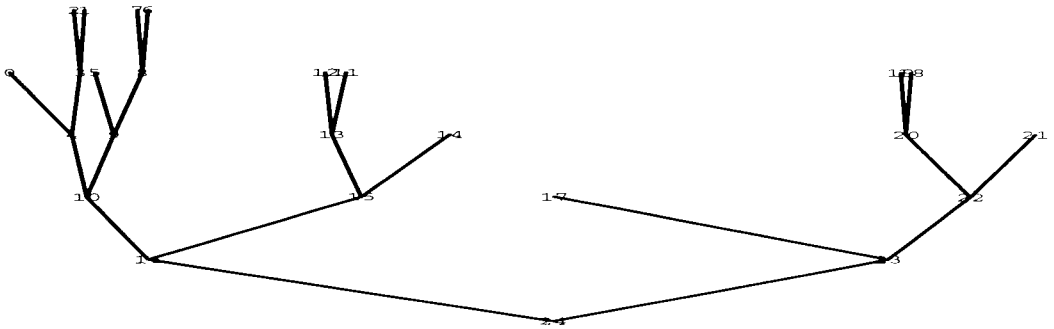


Node number	Data	Count
24	2.000000	23
23	2.000000	1
22	9.000000	22
21	9.000000	19
20	9.000000	15
19	9.000000	10
18	9.000000	5
17	10.000000	5
16	10.000000	3
15	15.000000	2
14	15.000000	1
13	14.000000	1
12	32.000000	5
11	32.000000	2
10	34.000000	3
9	34.000000	1
8	44.000000	2
7	44.000000	1
6	54.000000	1
5	100.000000	4
4	100.000000	2
3	87.000000	2
2	87.000000	1
1	88.000000	1
0	8.000000	3

Figure 4: A tree generated from artificial data

to the subtree root in order to update the usage counts in each node. This is a simple by-product of the recursive nature of the procedure.

A tree generated by a small set of artificial data is shown in figure 4. The LBG-based restructure algorithm was applied to the root node and the result is shown in figure 5. Note that the algorithm has produced a tree which represents the optimal clustering of the data at each node. This will prove useful in providing near-optimal bit rates when encoding data to a specified rate.



Node number	Data	Count
24	53.923077	23
23	85.800000	5
22	93.750000	4
21	100.000000	2
20	87.500000	2
19	87.000000	1
18	88.000000	1
17	54.000000	1
16	15.111111	18
15	35.500000	4
14	44.000000	1
13	32.666667	3
12	32.000000	2
11	34.000000	1
10	9.285714	14
9	11.800000	5
8	14.500000	2
7	14.000000	1
6	15.000000	1
5	10.000000	3
4	7.888889	9
3	8.625000	8
2	8.000000	3
1	9.000000	5
0	2.000000	1

Figure 5: The tree restructured from the root

Image	Search	Rate (bpp)	Compression Ratio	Peak-RMS SNR
Lena	Full	0.25	32:1	31.61 dB
Lena	None	0.20	40:1	31.52 dB
Boats	Full	0.34	23.5:1	29.95 dB
Boats	None	0.28	28.5:1	29.94 dB

Table 1: Coder test results

Figure 6: Reconstruction of Lena with no search

4 Coder Results

In this section, sample compressed and reconstructed images at low bit rates with and without searching on the standard *Lena* and *Boats* images are presented. The test images are 512 × 512 pixels in 8-bit greyscale. For the tests, a block size of 4 pixels square and the Daubechies 8 point filter are used and the Karhunen-Loève transform sub-system is enabled. The results of these tests are shown in the Table 1 and the following figures.

For comparison, Figure 10 shows the Lena image coded at 0.22 bits per pixel with the JPEG system. This image has a PSNR of 30.71 dB and much more visible artifacts. Shapiro's EZW coder [14] codes Lena at 0.25 bpp with a Peak-RMS SNR

Figure 7: Reconstruction of Lena with full search

Figure 8: Reconstruction of Boats with no search

Figure 9: Reconstruction of Boats with full search

of 33.17 dB and at 0.125 bpp with a PSNR of 30.23 dB. The coder presented here, achieves 31.52 dB at 0.20 bpp, a comparable figure.

As can be seen from the above results, at low bit rates, the search does not improve the coder performance. However, further tests have shown that at higher bit rates, a search is advantageous. Using a full search, the Lena image can be coded at 1.49 bpp with a PSNR of 38.67 dB. With no search, the image is coded in 1.44 bpp with a PSNR of 38.07 dB. At these bit rates, a gain of 0.6 dB for 0.05 bpp is significant.

Figure 10: JPEG Reconstruction at 0.22 bpp, PSNR 30.71

5 Tree Structured Vector Quantizer Results

In this section, results for the tree structured vector quantizer are presented. The quantizer is run on several sets of random data taken from Gaussian sources of different variances in one dimension. The results are compared with the theoretical rate, r_d, for encoding random data from a Gaussian distribution with variance σ^2 at distortion d.

$$r_d = \log_2 \left(\frac{\sigma^2}{d} \right) \tag{16}$$

The quantizer is presented with 32768 points taken from a Gaussian distribution of variance σ^2. Data points are rounded to the nearest integer (to allow multiple data in leaf nodes). Once the tree has been built, the training data is presented to be coded at various allowable distortions. The average rate is recorded and compared with the theoretical value. Also, the actual average error is reported. Then, the tree is restructured (if necessary) and the same data applied again. Finally, ten random random sets of data from outside the training set are applied and the results averaged. This is repeated for a variety of variances on the underlying Gaussian distribution.

Note that the minimum bit rate for any particular vector is 1 bit. This is simply because the coder must emit at least a single bit to indicate that a vector has arrived.

Max Error	Actual Rate	Theoretical Rate	Actual Error
0	7.57724	9.96578	0
50	3.56409	4.32193	25.9154
100	3.46811	3.32193	29.313
150	3.4006	2.73697	32.1116
200	3.09796	2.32193	52.2774
250	2.96152	2	64.9643
300	2.66391	1.73697	95.3608
350	2.66391	1.51457	95.3608
400	2.66391	1.32193	95.3608
450	2.27911	1.152	195.78
500	2.27911	1	195.78
550	1.60513	0.862496	373.108
600	1.60513	0.736966	373.108
650	1.60513	0.621488	373.108
700	1.60513	0.514573	373.108
750	1.60513	0.415037	373.108
800	1.60513	0.321928	373.108
850	1.60513	0.234465	373.108
900	1	0.152003	735.434
950	1	0.0740006	735.434
1000	1	0	735.434

Table 2: Results on training set before LBG restructure $\sigma^2 = 1000$

Max Error	Actual Rate	Theoretical Rate	Actual Error
0	7.20197	9.96578	0
50	3.14124	4.32193	23.1866
100	2.47821	3.32193	58.3728
150	2.34482	2.73697	69.9259
200	2.34482	2.32193	69.9259
250	2	2	117.279
300	2	1.73697	117.279
350	2	1.51457	117.279
400	1	1.32193	361.154
450	1	1.152	361.154
500	1	1	361.154
550	1	0.862496	361.154
600	1	0.736966	361.154
650	1	0.621488	361.154
700	1	0.514573	361.154
750	1	0.415037	361.154
800	1	0.321928	361.154
850	1	0.234465	361.154
900	1	0.152003	361.154
950	1	0.0740006	361.154
1000	1	0	361.154

Table 3: Results on training set after LBG restructure $\sigma^2 = 1000$

Max Error	Actual Rate	Theoretical Rate	Actual Error
0	7.20871	9.96578	0.00557251
50	3.14101	4.32193	23.1801
100	2.47671	3.32193	58.6911
150	2.34391	2.73697	70.3585
200	2.34391	2.32193	70.3585
250	2	2	117.863
300	2	1.73697	117.863
350	2	1.51457	117.863
400	1	1.32193	363.283
450	1	1.152	363.283
500	1	1	363.283
550	1	0.862496	363.283
600	1	0.736966	363.283
650	1	0.621488	363.283
700	1	0.514573	363.283
750	1	0.415037	363.283
800	1	0.321928	363.283
850	1	0.234465	363.283
900	1	0.152003	363.283
950	1	0.0740006	363.283
1000	1	0	363.283

Table 4: Results outside training set after LBG restructure $\sigma^2 = 1000$

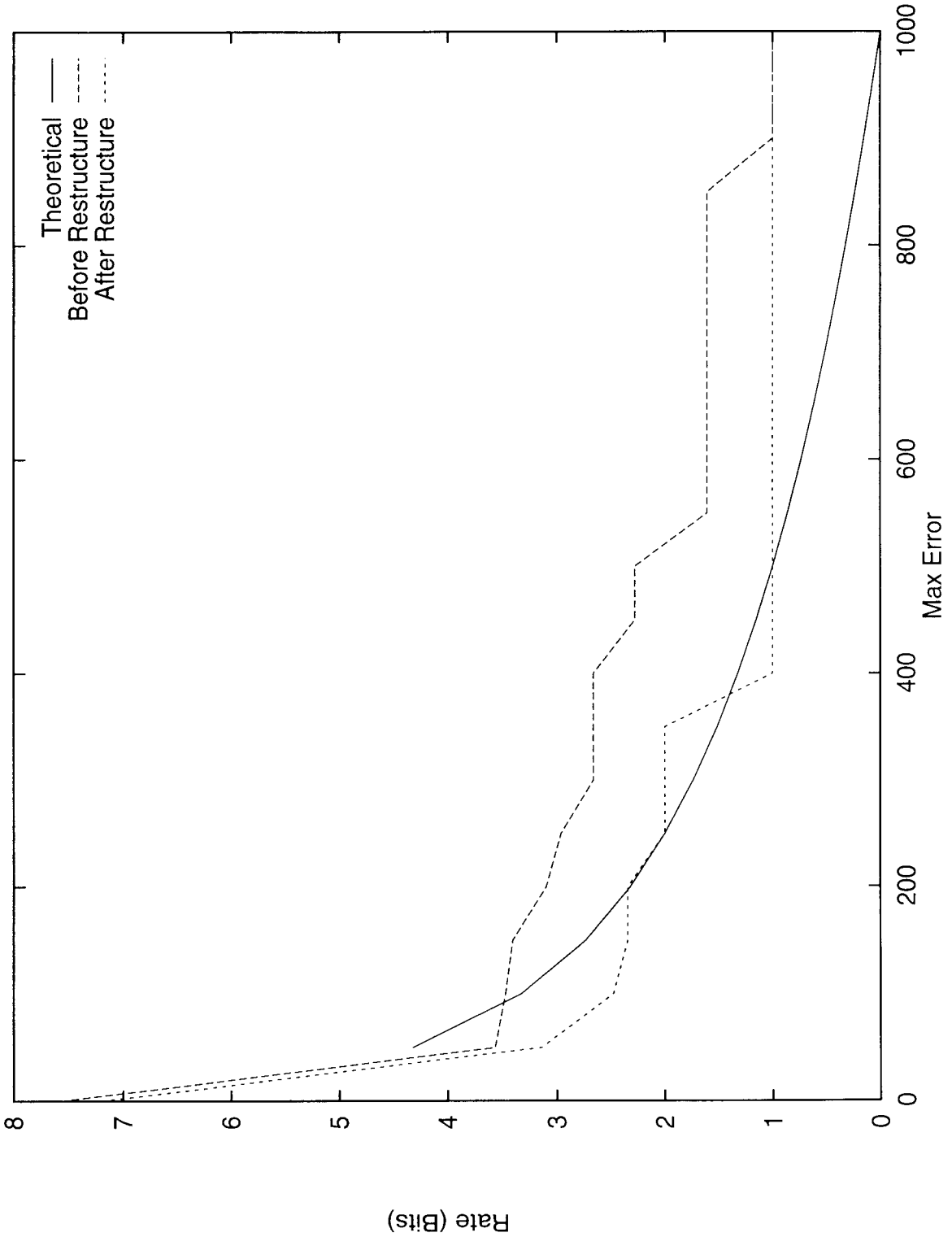


Figure 11: Results for data in the training set $\sigma^2 = 1000$

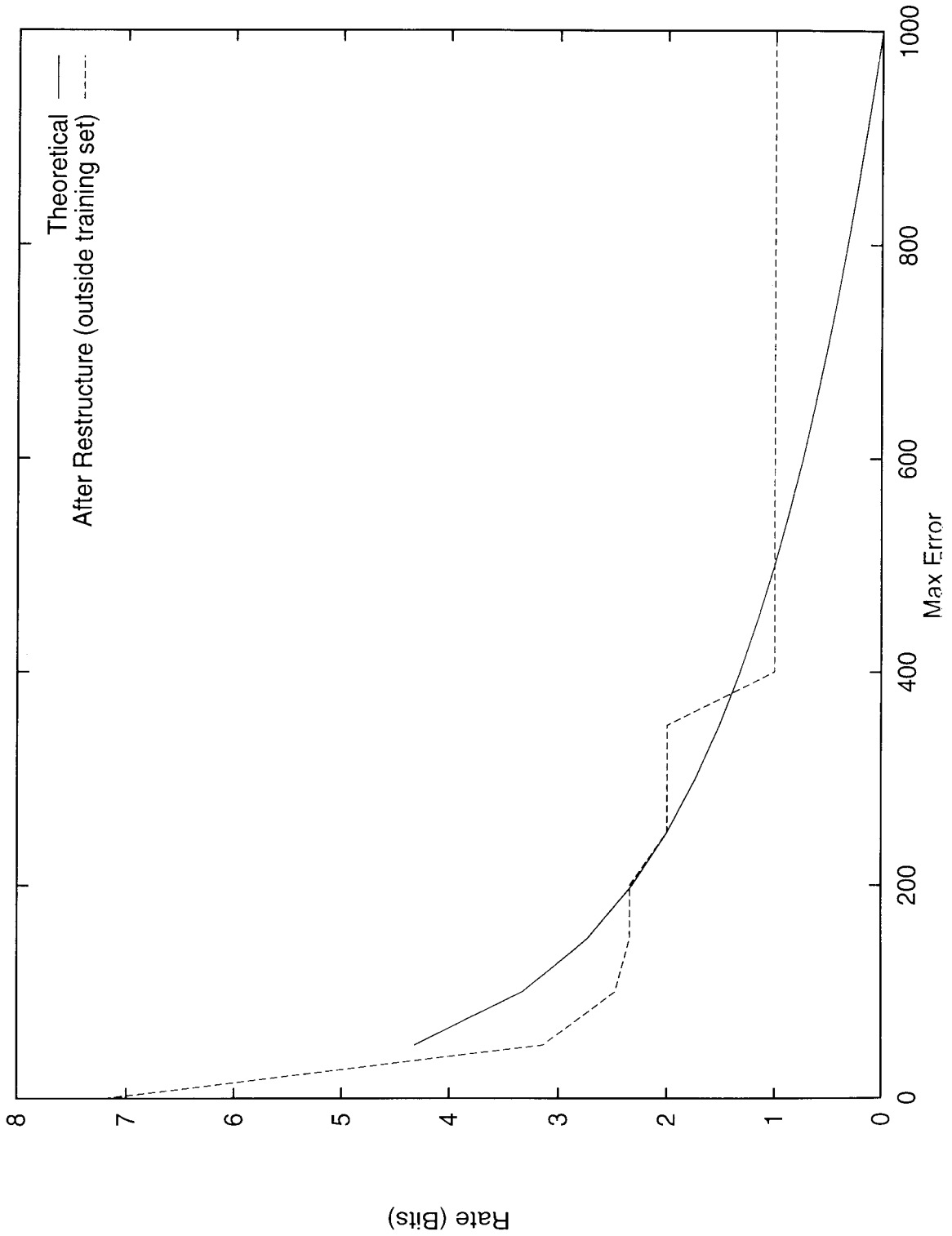


Figure 12: Results for data outside training set $\sigma^2 = 1000$

Max Error	Actual Rate	Theoretical Rate	Actual Error
0	7.11093	7.64386	0
10	4.64316	4.32193	3.25531
20	4.5694	3.32193	3.86703
30	3.85818	2.73697	17.9718
40	3.76474	2.32193	19.4426
50	3.76474	2	19.4426
60	3.76474	1.73697	19.4426
70	3.65417	1.51457	24.2237
80	3.61078	1.32193	27.0661
90	2.96503	1.152	40.8026
100	2.96503	1	40.8026
110	2.59341	0.862496	56.8199
120	2.59341	0.736966	56.8199
130	1.88943	0.621488	98.6811
140	1.88943	0.514573	98.6811
150	1.88943	0.415037	98.6811
160	1.88943	0.321928	98.6811
170	1.88943	0.234465	98.6811
180	1.88943	0.152003	98.6811
190	1.88943	0.0740006	98.6811
200	1.88943	0	98.6811

Table 5: Results on training set before LBG restructure $\sigma^2 = 200$

Max Error	Actual Rate	Theoretical Rate	Actual Error
0	6.0369	7.64386	0
10	3.15997	4.32193	4.40185
20	2.51111	3.32193	10.9694
30	2.35965	2.73697	13.6658
40	2.35965	2.32193	13.6658
50	2	2	23.8173
60	2	1.73697	23.8173
70	2	1.51457	23.8173
80	1	1.32193	73.7492
90	1	1.152	73.7492
100	1	1	73.7492
110	1	0.862496	73.7492
120	1	0.736966	73.7492
130	1	0.621488	73.7492
140	1	0.514573	73.7492
150	1	0.415037	73.7492
160	1	0.321928	73.7492
170	1	0.234465	73.7492
180	1	0.152003	73.7492
190	1	0.0740006	73.7492
200	1	0	73.7492

Table 6: Results on training set after LBG restructure $\sigma^2 = 200$

Max Error	Actual Rate	Theoretical Rate	Actual Error
0	6.04124	7.64386	0.000714111
10	3.15867	4.32193	4.41232
20	2.50854	3.32193	10.9812
30	2.35868	2.73697	13.666
40	2.35868	2.32193	13.666
50	2	2	23.7635
60	2	1.73697	23.7635
70	2	1.51457	23.7635
80	1	1.32193	73.0034
90	1	1.152	73.0034
100	1	1	73.0034
110	1	0.862496	73.0034
120	1	0.736966	73.0034
130	1	0.621488	73.0034
140	1	0.514573	73.0034
150	1	0.415037	73.0034
160	1	0.321928	73.0034
170	1	0.234465	73.0034
180	1	0.152003	73.0034
190	1	0.0740006	73.0034
200	1	0	73.0034

Table 7: Results outside training set after LBG restructure $\sigma^2 = 200$

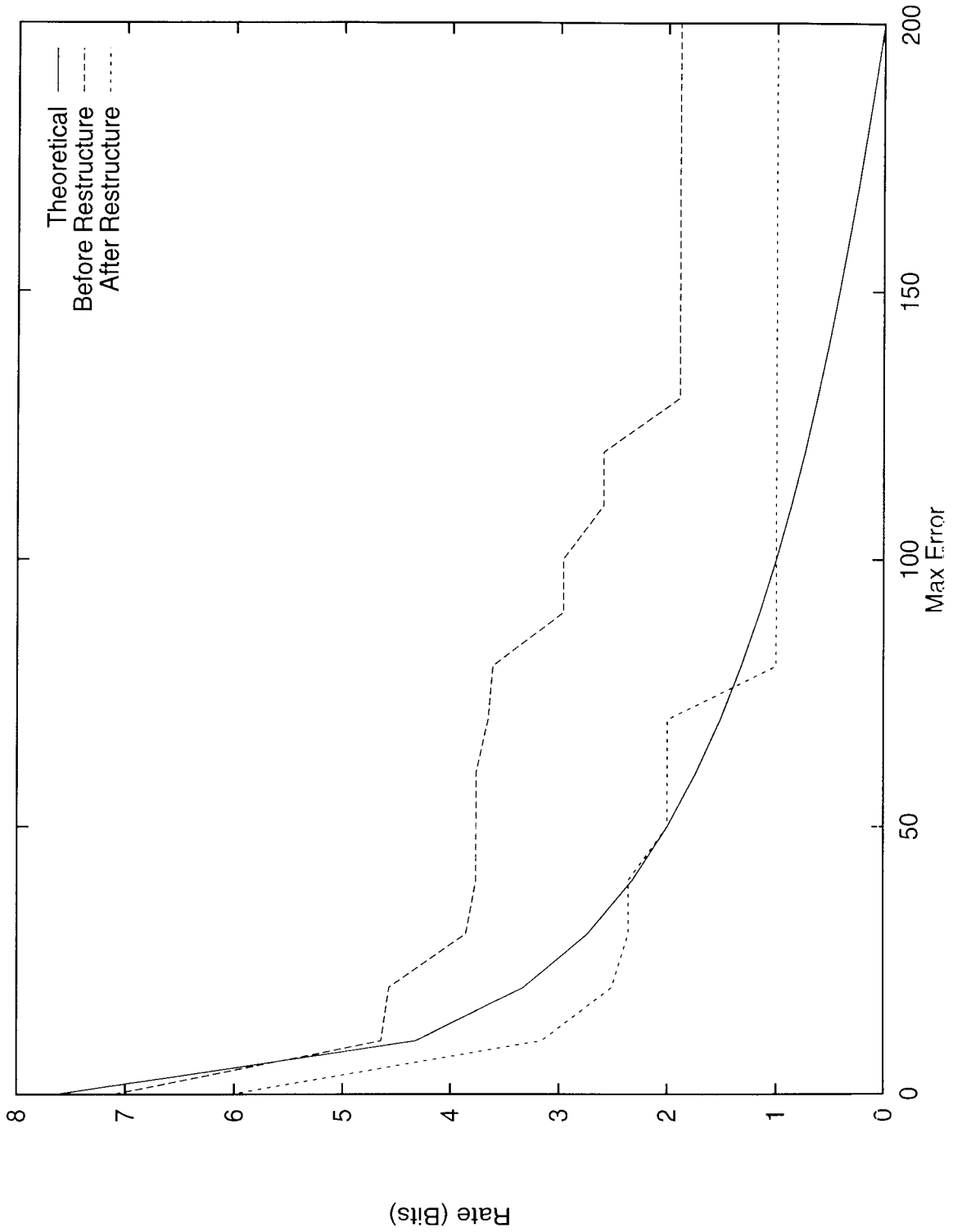


Figure 13: Results for data in the training set $\sigma^2 = 200$

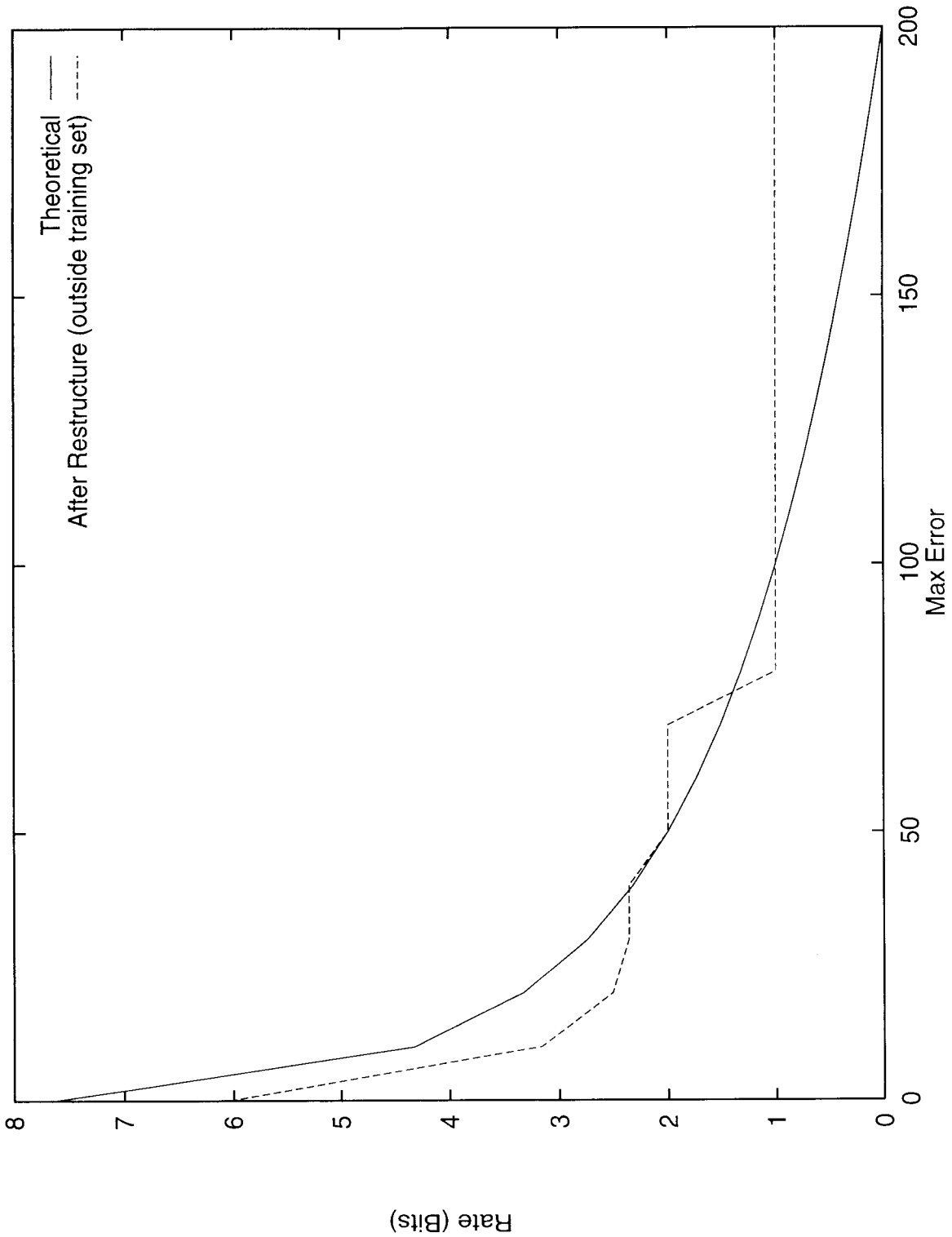


Figure 14: Results for data outside training set $\sigma^2 = 200$

6 Discussion,Conclusions and Further Work

This paper has shown that it is possible to place fractal coding into a multiresolution predictive framework by using an orthonormal wavelet transform and a novel block searching technique. Initial results from the coder show that the method has real promise in applications. Also, a novel tree structured vector quantizer has been presented which exhibits near optimal rate-distortion performance. One significant problem with this technique is the excessive search time. For the highest quality (ie a full search), coding time is prohibitive. However, integrating the tree structured search should help in this matter. Another area requiring more work is the choice of wavelets - the lack of symmetry of the separable Daubechies wavelets clearly has an impact on the quality of the results. A wavelet packet type basis could be employed to aid orientation matching.

Finally, the coder could be extended to operate in 3 dimensional space, that is video sequences. The tree structured vector quantizer would provide an ever growing pool of blocks to be used for prediction between frames and it could be relatively quickly restructured when required. This would hopefully exploit the redundancy between frames at very low bit cost.

References

[1] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using Wavelet Transform. In *IEEE Transactions on Image Processing*, volume 1, 1992.

[2] M. Balakrishnan, W. Pearlman, and L. Lu. Variable-Rate Tree-Structured Vector Quantizers. In *IEEE Transactions on Information Theory*, volume 41, 1995.

[3] M. Barnsley and S. Demko. Iterated Function Systems and the Global Construction of Fractals. In *Proc. Royal Society of London*, 1985.

[4] M. Barnsley and L. Hurd. *Fractal Image Compression.* AK Peters, 1992.

[5] K. Barthel and T. Voyé. Adaptive Fractal Image Coding in the Frequency Domain. In *Proceedings of International Workshop on Image Processing*, 1994.

[6] M. Gharavi-Alkhansari and T. Huang. Fractal-Based Techniques for a Generalized Image Coding Method. In *Proc. IEEE ICASSP*, 1994.

[7] A. E. Jacquin. A Novel Fractal Block-Coding Technique for Digital Images. In *Proc. ICASSP '90*, 1990.

[8] I. Levy and R. Wilson. A Hybrid Fractal-Wavelet Transform Image Data Compression Algorithm. Technical Report 289, University of Warwick, Department of Computer Science, September 1995.

[9] I. Levy and R. Wilson. Predictive Wavelet Transform Coding : Unifying Fractal and Transform coding. In *Proceedings PCS96*, 1996.

[10] Y. Linde, A. Buzo, and R Gray. An Algorithm for Vector Quantizer Design. In *IEEE Transactions on Communications*, volume 28, 1980.

[11] D. Monro and F. Dudbridge. Fractal Approximation of image Blocks. In *Proc. IEEE ICASSP*, 1992.

[12] L-M. Po and C-K. Chan. Adaptive Dimensionality Reduction Techniques for Tree-structured Vector Quantization. In *IEEE Transactions on Communications*, volume 42, 1994.

[13] R. Rinaldo and G. Calvagno. Image Coding by Block Prediction of Multiresolution Subimages. Technical report, Universitá di Padova, 1995.

[14] J. Shapiro. Embedded Image Coding Using Zerotrees of Wavelet Coefficients. In *IEEE Transactions on Signal Processing*, volume 41, 1993.